**Nextjs functions**

**Cookies**

The cookies function allows us to read http incoming request cookies from a server component / write outgoing request cookies in a server action / route handler

cookies().get (name)

A method that takes a cookie name and returns object with name and value. If a cookie name isnt found, it returns **undefined**. If multiple cookies match, it will only return the first match.

cookies().getAll()

A method that is similar to get, but returns a list of all the cookies with a matching name. if name is unspecified, it returns all the available cookies.

cookies().has(name)

A method that takes a cookie name and return a Boolean based on if the cookie exist(true) or not (false)

cookies().set(name,value,options)

A method that takes a cookie name, value, and options and sets the outgoing request cookie.

Removing cookies

We can only remove cookies in a server action / route handler

cookie().delete(name)

we can explicitly remove a cookie with a given name

cookies().set(name,””)

alternatively, we can set a new cookie with the same name and empty value.

cookies().set(name, value, {maxAge: 0})

setting maxAge to 0 will immediately expire a cookie

cookies().set(name,value,{expires: timestamp})

setting expires to any value in the past will immediately expire a cookie.

**useParams**

useParams is a client component hook that lets you read a routes dynamic params filled in by the current url

'use client'

import { useParams } from 'next/navigation'

const params = useParams()

useParams return object that containing the current routes dynamic parameter.

Each property in the object is active dynamic segment

The property name is the segment’s name and the properties value is what the property filled with.

The property value will either be a string or array of string depending on the type of dynamic segment

If route contains no dynamic parameter, useParams returns a empty object.

|  |  |  |
| --- | --- | --- |
| **Route** | **URL** | **useParams()** |
| app/shop/page.js | /shop | {} |
| app/shop/[slug]/page.js | /shop/1 | {slug:”1”} |
| app/shop/[tag]/[item]/page.js | /shop/1/2 | {tag:1,item:2} |
| app/shop/[…slug]/page.js | /shop/1/2 | {slug:[“1”,”2”]} |

**usePathname**

usePathname is a client component hook that lets you read the current url pathname. In a client component with usePathname will be rendered into HTML on the initial page load. When navigating to a new route, this component does not need to be refetched.

'use client'

import { usePathname } from 'next/navigation'

const pathname = usePathname()

Reading the current url from a server component is not supported. This design is intentional to support layout state being preserved across page navigation.

Compatibility mode

usePathname can return null when a fallback route is being rendered or when a page directory page has been automatically statically optimized by nextjs and the router is not ready.

Nextjs will automatically update our types if it detects both app and pages directory in our project.

usePathname returns a string of the current url pathname

|  |  |
| --- | --- |
| **URL** | **Returned value** |
| / | ‘/’ |
| /dashboard | “/dashboard” |
| /dashboard?v=2 | “/dashboard” |
| /blog/server-rendering | “/blog/server-rendering” |

**useRouter**

The useRouter hook allows us to programmatically change route inside client component. Use the <Link> component for navigation unless we have a specific requirement for using useRouter.

'use client'

import { useRouter } from 'next/navigation'

const router = useRouter()

router.push (href: string, {scroll: boolean})

By using this method we can insert new route / navigate to provided route. It add a new entry into the browser history.

router.replace (href: string, {scroll: boolean}

By using this we can redirect / navigate to provided route without adding a new entry into the browser history stack.

router.refresh ()

By using this we are refreshing the current route. Making new request to the server, re-fetching data requests, and re-rendering server components.

The client will merge the updated react server component payload without losing unaffected client-side react or browser state.

router.prefetch (href: string)

Prefetch the provided route for faster client-side transitions.

Prefetching is a way to preload a route in the background before the user visits it.

There are two ways routes are prefetched in nextjs

<Link> component

router.prefetch ()

router.back ()

navigate back to the previous route in the browser history stack.

router.forward ()

navigate forwards to the next page in the browser history stack

Migrating from next/router

The **useRouter hook** should be imported from **next/navigation** and not **next/router** when using the app router.

The pathname string has been removed and is replaced by usePathname ()

The query object has been removed and is replaced by useSearchParams ()

router.events has been replaced.

**Router events**

We can listen for page changes by composing other client component hooks like **usePathname** and **useSearchParams**.

"use client";

import { useEffect } from "react";

import { usePathname, useSearchParams } from "next/navigation";

import { useSearchParams } from "react-router-dom";

export function NavigationEvents() {

  const pathname = usePathname();

  const searchParams = useSearchParams();

  useEffect(() => {

    const url = `${pathname}?${searchParams}`;

    return url;

  }, [pathname, searchParams]);

  return null;

}

import { Suspense } from "react";

import { NavigationEvents } from "./components/navigation-events";

export default function Layout({ children }) {

  return (

    <html lang="en">

      <body>

        {children}

        <Suspense fallback={null}>

          <NavigationEvents />

        </Suspense>

      </body>

    </html>

  );

}

Disabling scroll restoration

By default, nextjs will scroll to the top of the page when navigating to a new route. We can disable this behavior by passing scroll:false to router.push() / router.replace()